GDSC AI Week
1st Day

고려대학교 정보대학
Korea University
College of Informatics

전병우
ipcs@korea.ac.kr

Team Building

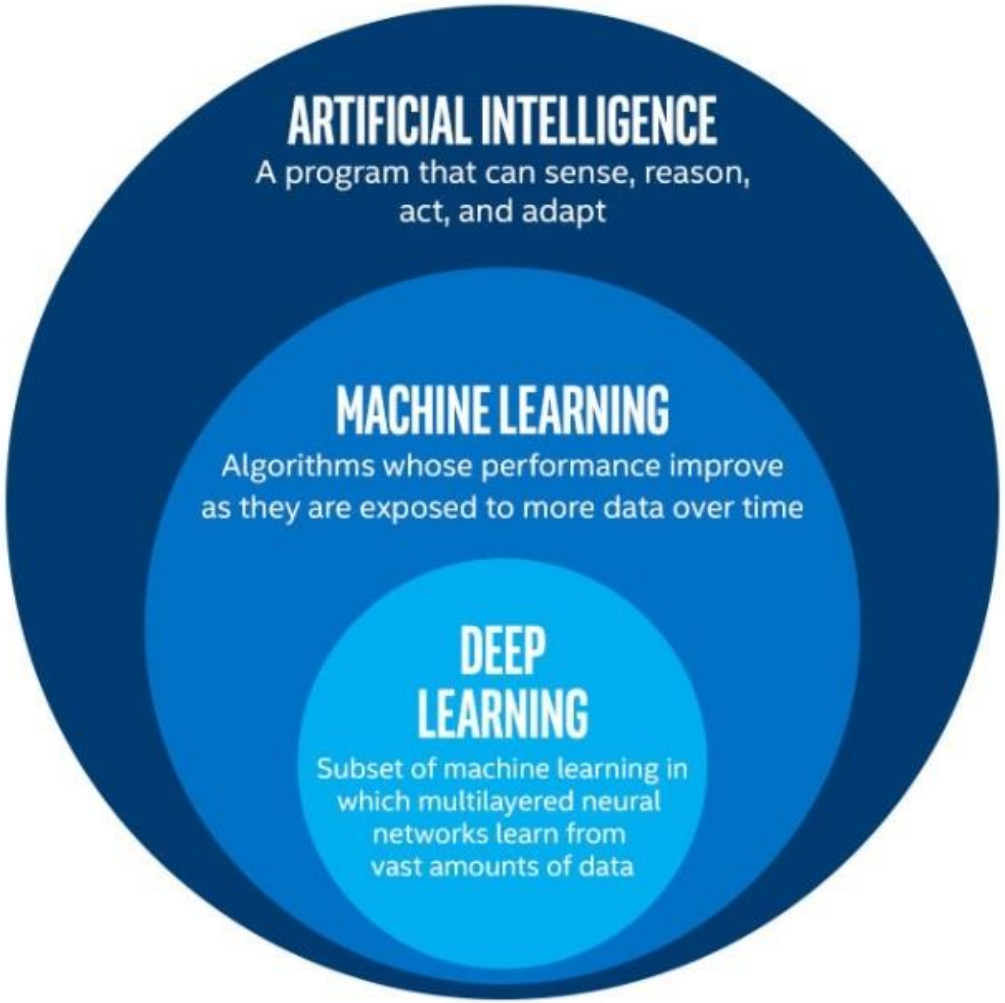
Team A	Team B	Team C	Team D
고수영	강성은	이지윤	장형석
김태우	김서영	원다혜	조혜원
송보미	김태관	김규민	도민욱(P)
이예일(P)	이유찬(P)	김모세(P)	

Hello, Everyone! I am..

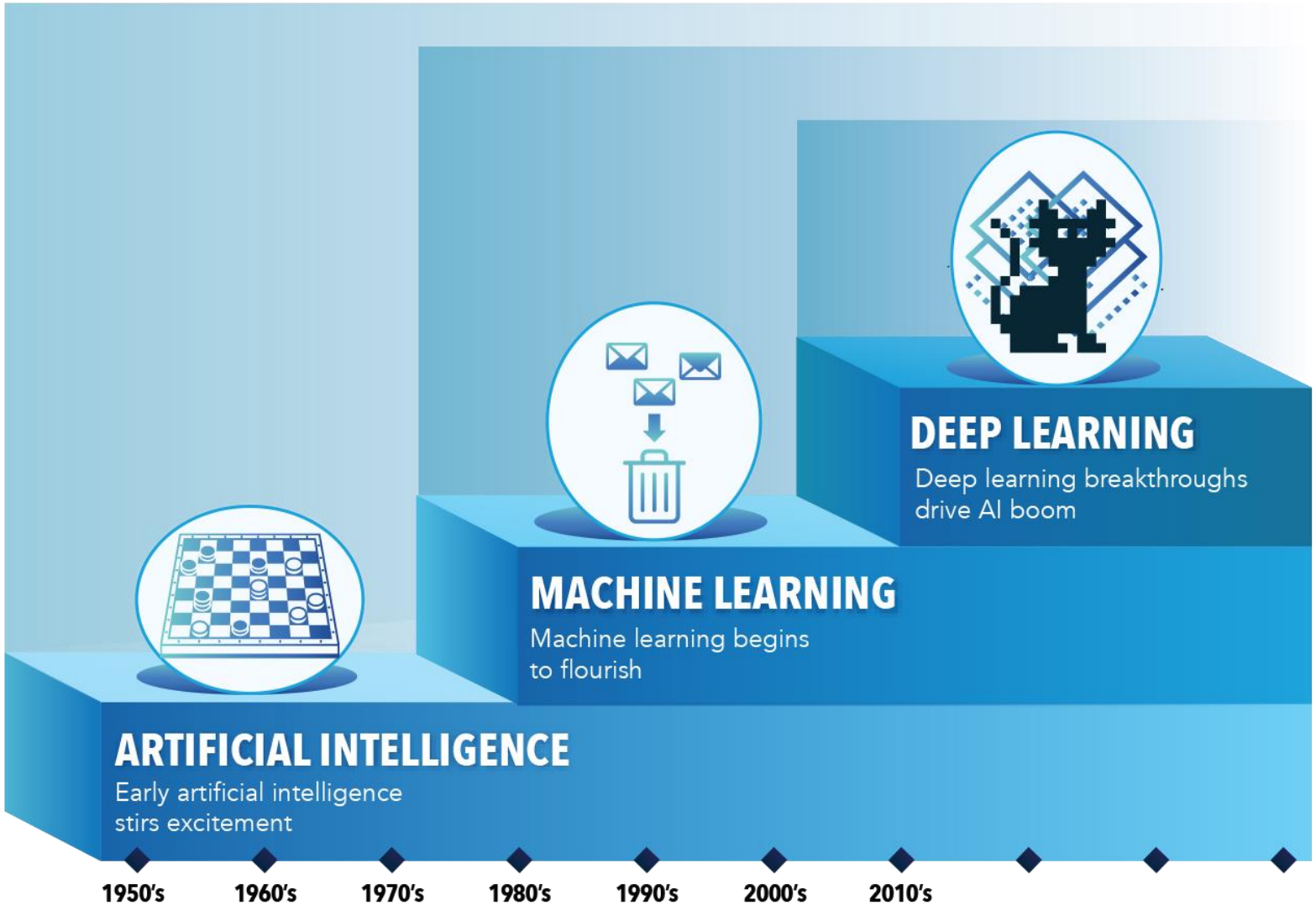
전병우

- 고려대학교 컴퓨터학과, 통계학과(이중) (20.03 ~ 24.08)
- 동아리/학회
  - GDSC KU 1st Lead (22.08 ~ 23.08)
  - GDSC KU 2nd AI Core (23.09 ~ Present)
  - KUBIG (23.08 ~ Present)
- 연구
  - ALINLAB, 카이스트 AI (신진우 교수님) (23.01 ~ Present)
  - LIMLAB, 고려대 (임성빈 교수님) (23.09 ~ Present)
  - MLVLAB, 고려대 (김현우 교수님) (22.08 ~ 22.12)
- 회사
  - ARCREAL Co. (23.01 ~ Present)
  - M-Monstar (21.07 ~ 21.08)
- Blog: https://rootyjeon.github.io/
- Github: https://github.com/rootyJeon

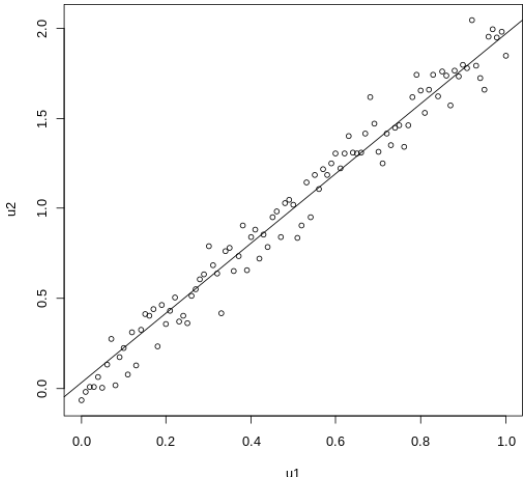
What is an AI?

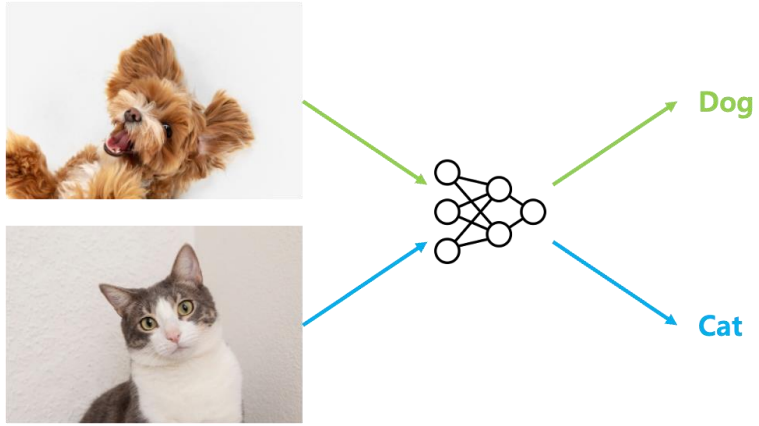

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

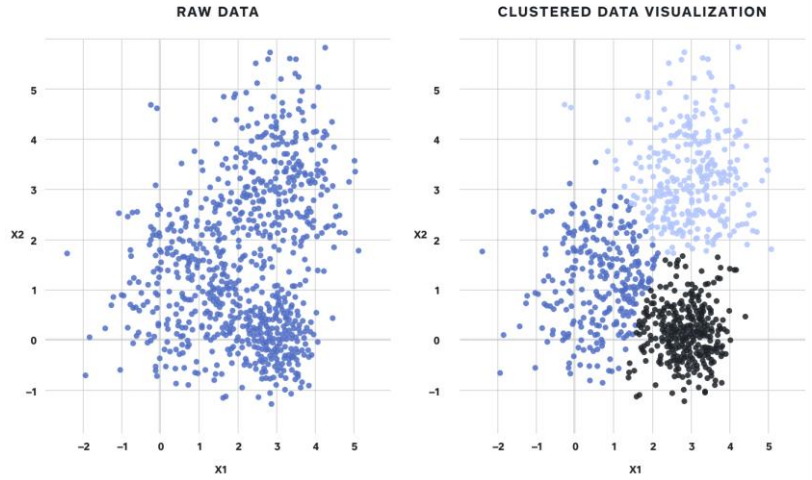
History of AI



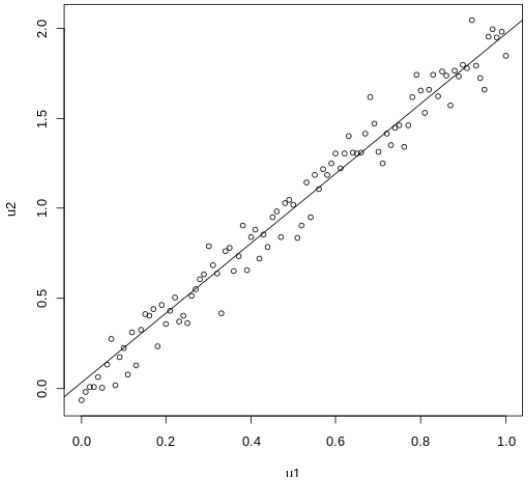
What problems AI can solve?



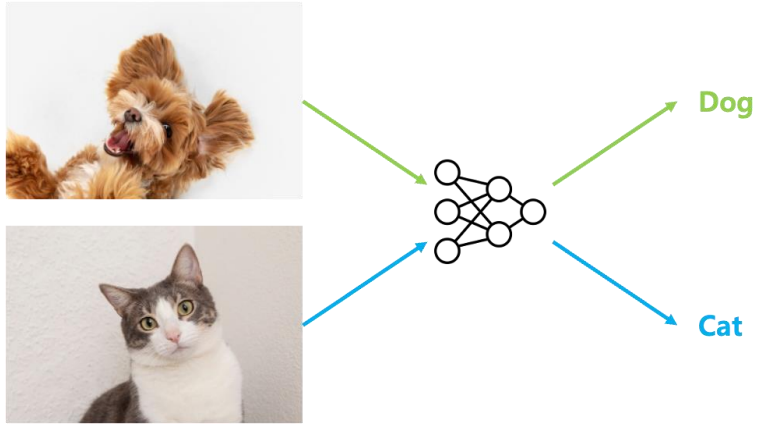




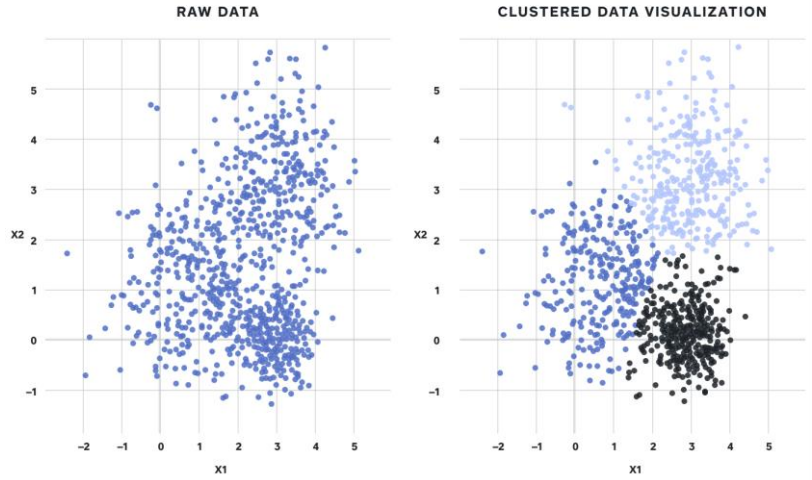
What problems AI can solve?



Regression

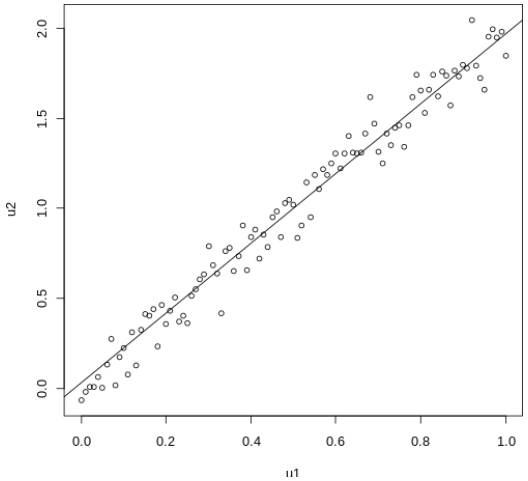


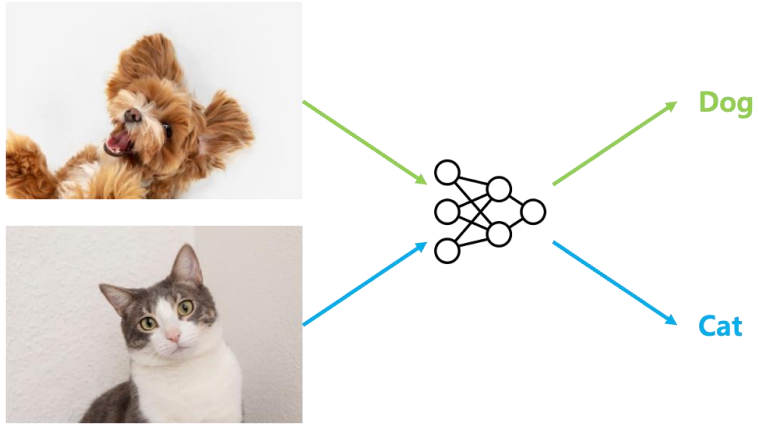
Classification

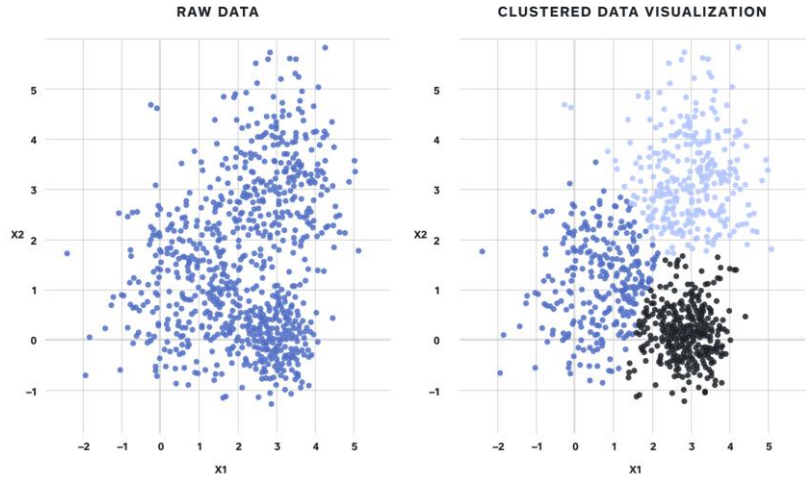


Clustering

What problems AI can solve?







Regression Classification

Clustering

Supervised-Learning

Unsupervised-Learning

First, you will learn supervised learning because this is a very intuitive task!

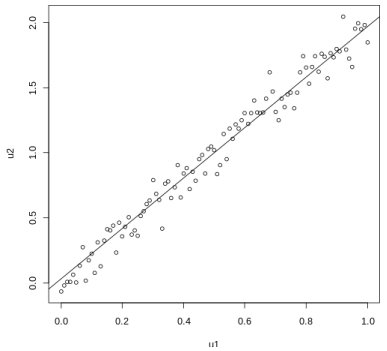
Machine Learning Tasks

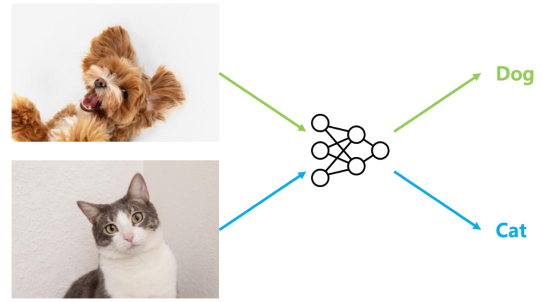
- Supervised Learning
  - You need labels (Ground Truth)
  - Classification, Regression

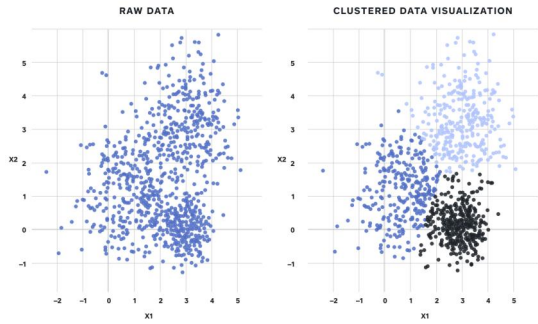
- Unsupervised Learning
  - You don't use labels
  - Clustering, GAN

- Reinforcement Learning
  - You need to define Agent, Env, State, Action, and Rewards
  - AlphaGO

- Note that these are Machine Learning tasks.
- Deep learning can be used to solve all of them!







Example: Linear Regression

$$y = wx + b + \varepsilon$$

Example: Linear Regression

label

feature

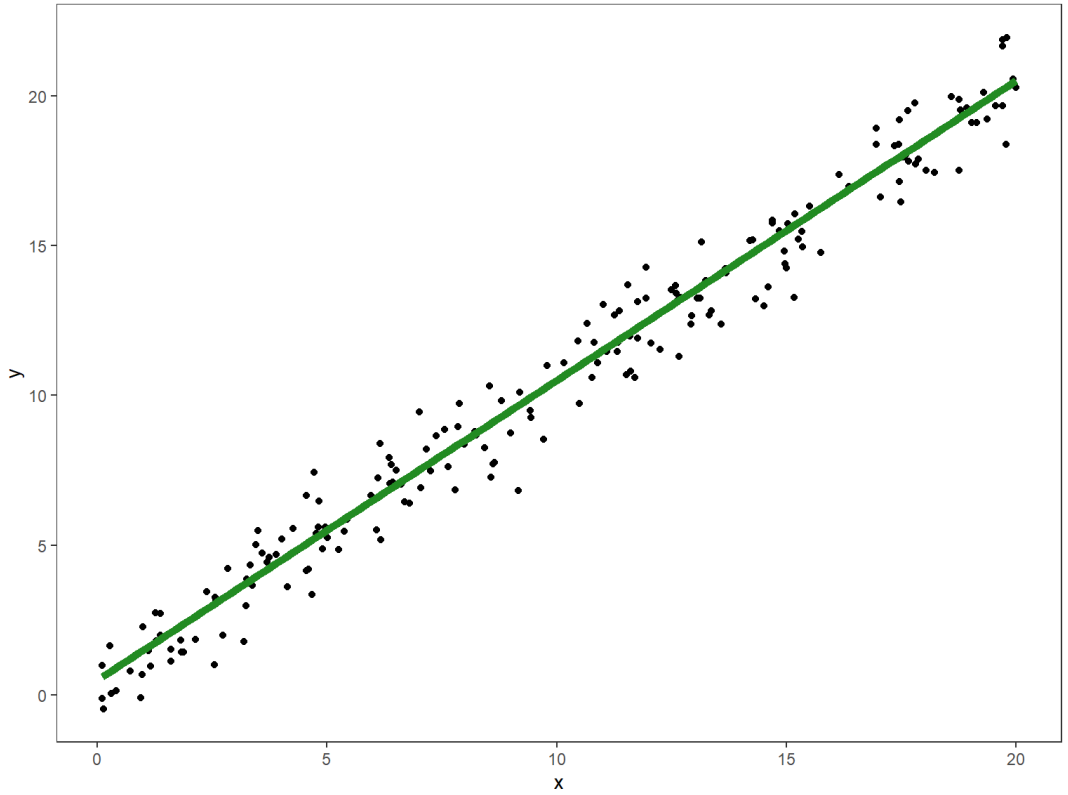
$$y = wx + b + \varepsilon$$

weight(unknown)

weight(unknown)

noise (not controllable)

What is the Noise?

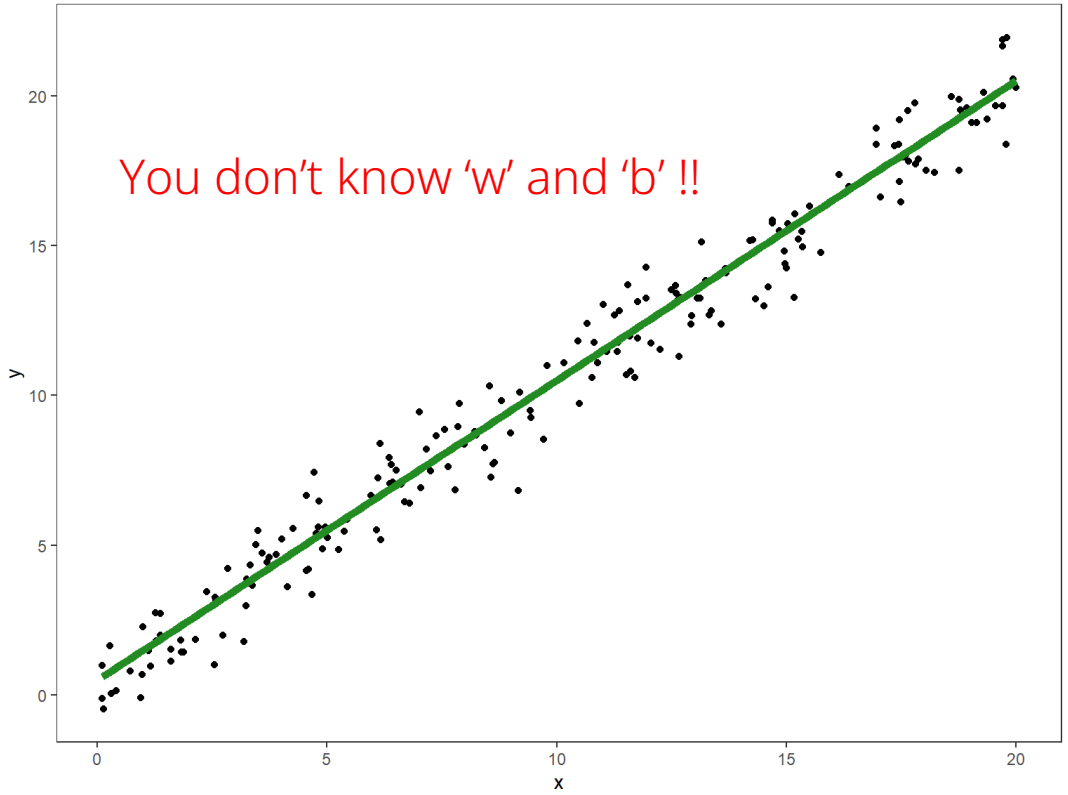


feature

label $y = wx + b + \varepsilon$

weight(unknown) weight(unknown) noise (not controllable)

Example: Linear Regression



feature

label $y = wx + b + \varepsilon$

weight(unknown) weight(unknown) noise (not controllable)

Example: Linear Regression

Label $$y = wx + b + \varepsilon$$

Estimate $$\hat{y} = \hat{w}x + \hat{b}$$

Example: Linear Regression

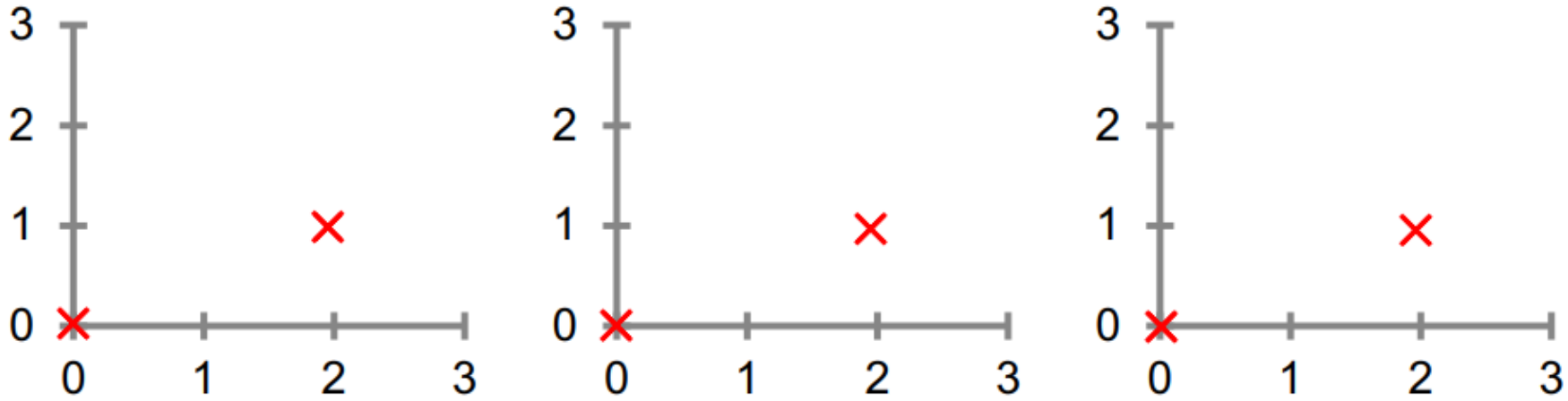
Label $$y = wx + b + \varepsilon$$

Estimate $$\hat{y} = \hat{w}x + \hat{b}$$

How can we get $\hat{w}$ and $\hat{b}$?

Example: Linear Regression

- 다음과 같은 선형 회귀 모델을 가정하자: $\widehat{y_n} = f_w(x_n) = \widehat{w_0} + \widehat{w_1} x_n$



$w_0 = 1.5$
$w_1 = 0$

$w_0 = 0$
$w_1 = 0.5$

$w_0 = 1$
$w_1 = 0.5$

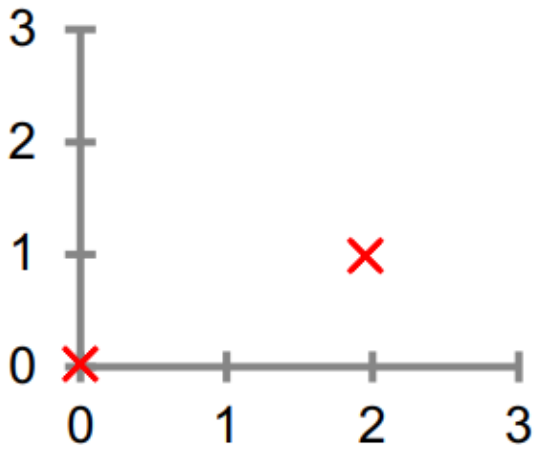
Mean Square Error (MSE)

- 우리는 y (label) 과 $\hat{y}$ (our prediction)의 거리를 최소화시키고 싶다.
- 그 전에 먼저 거리(distance)를 정의해야 한다.
- 거리를 차이의 제곱으로 정의하고 이것의 평균을 구하는 방법을 MSE 또는 L2 norm이라고 한다.

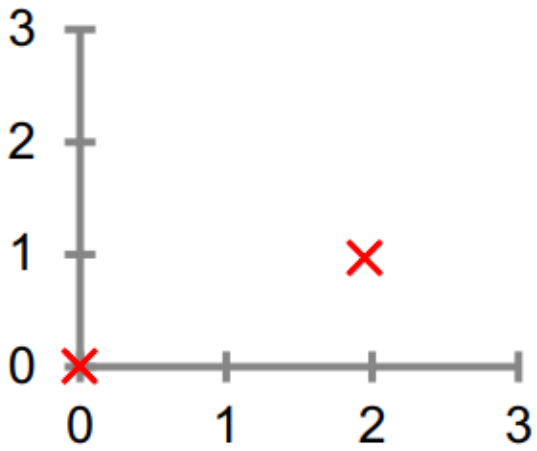
$$MSE(\hat{w}, \hat{b}) = \|y - \hat{y}\|_2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Mean Square Error (MSE)

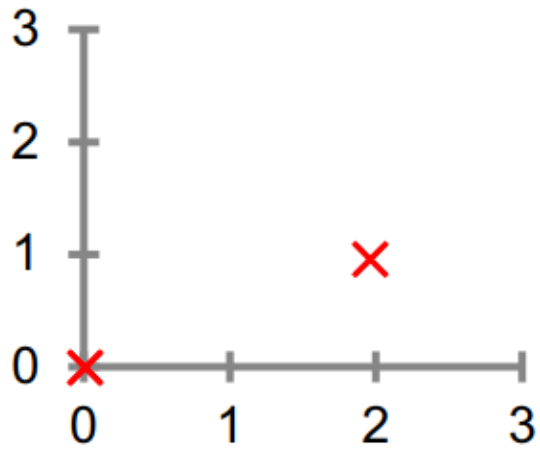
$$MSE(\hat{w}, \hat{b}) = \|y - \hat{y}\|_2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



$w_0 = 1.5$
$w_1 = 0$



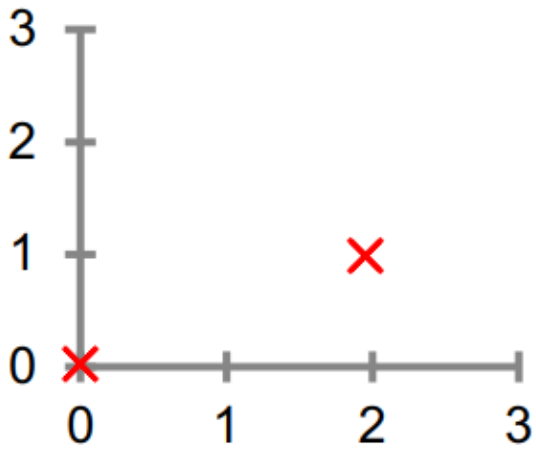
$w_0 = 0$
$w_1 = 0.5$



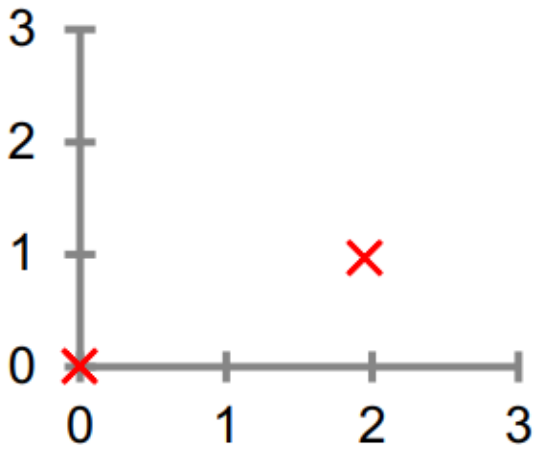
$w_0 = 1$
$w_1 = 0.5$

Learning Linear Regression

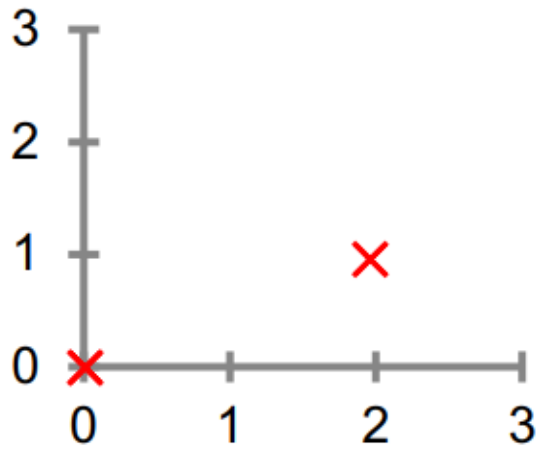
$$\underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$



$w_0 = 1.5$
$w_1 = 0$



$w_0 = 0$
$w_1 = 0.5$



$w_0 = 1$
$w_1 = 0.5$

거리(distance)를 다르게 정의할 수는 없을까?

Mean Absolute Error (MAE)

- 거리를 차이의 절댓값으로 정의하고 이것의 평균을 구하는 방법을 MAE 또는 L1 norm이라고 한다.

$$MAE(\hat{w}, \hat{b}) = \|y - \hat{y}\|_1 = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Loss function

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = \underset{w}{\operatorname{argmin}}\, L(w)$$

Define the machine learning model

- Model:
  - $\widehat{y_n} = f_w(x_n) = \widehat{w_0} + \widehat{w_1} x_n$
- Parameters:
  - $w = (w_0, w_1)$
- Loss function:
  - $L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{y_i})^2$
- Goal:
  - $w^* = \underset{w}{\operatorname{argmin}}\, L(w)$

Define the machine learning model

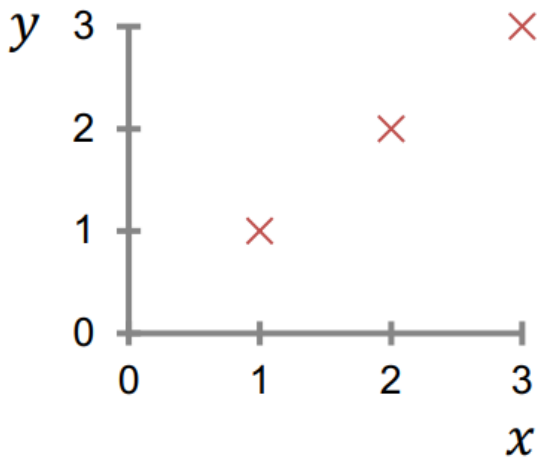
- Model:
  - $\widehat{y_n} = f_w(x_n) = \widehat{w_0} + \widehat{w_1} x_n$ 간단하게 하기 위해, bias(b)가 0이라고 하자!
- Parameters:
  - $w = (w_0, w_1)$
- Loss function:
  - $L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Goal:
  - $w^* = \underset{w}{\operatorname{argmin}}\, L(w)$

Define the machine learning model

- Model:
  - $\widehat{y_n} = f_w(x_n) = \widehat{w_0} + \widehat{w_1} x_n$
- Parameters:
  - $w = (w_0, w_1)$
- Loss function:
  - $L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Goal:
  - $w^* = \underset{w}{\operatorname{argmin}}\, L(w)$

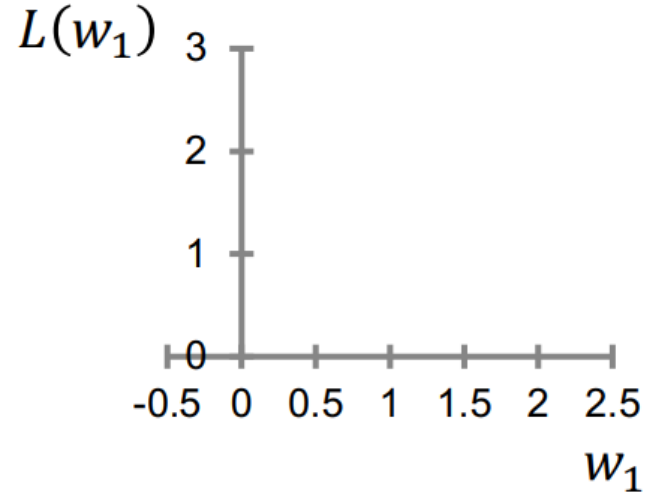
$f_w(x)$

(for fixed w_1, this is a function of x)



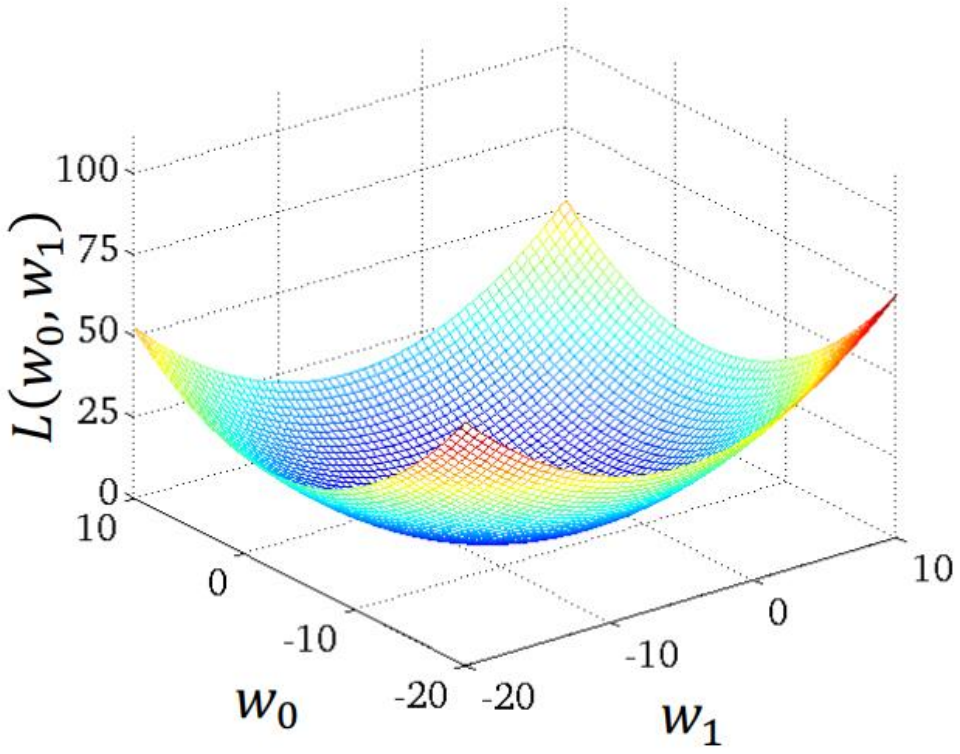
$L(w_1)$

(function of the parameter w_1)



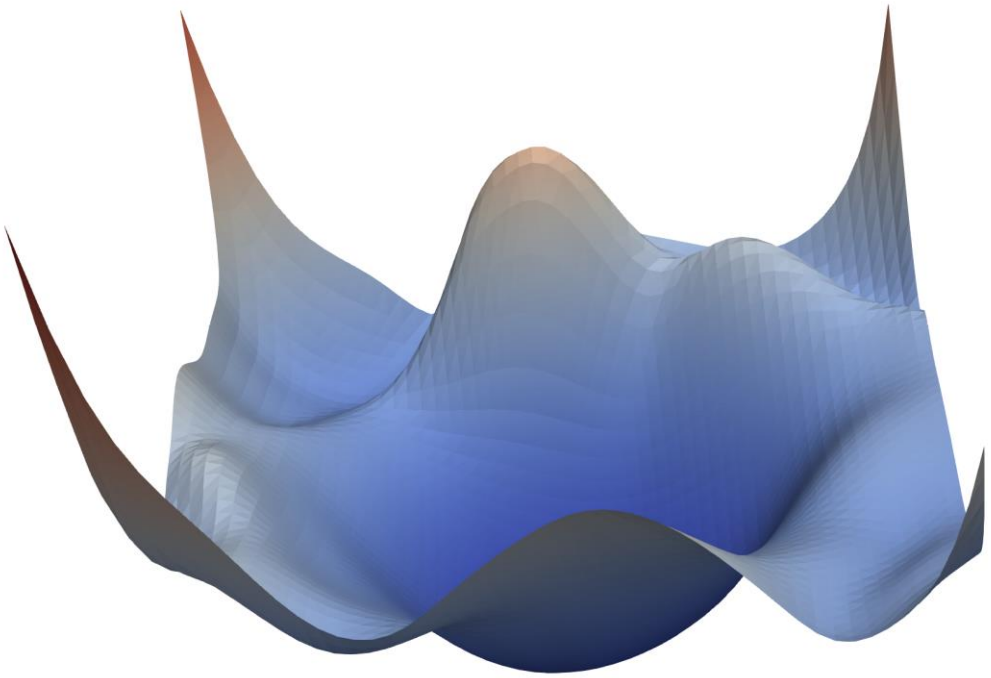
만약 우리의 모델이 다음처럼 복잡하다면? : $\widehat{y_n} = f_w(x_n) = \widehat{w_0} + \widehat{w_1} x_{1_n} + \widehat{w_2} x_{2_n} + \widehat{w_3} x_{3_n}$

Loss surface



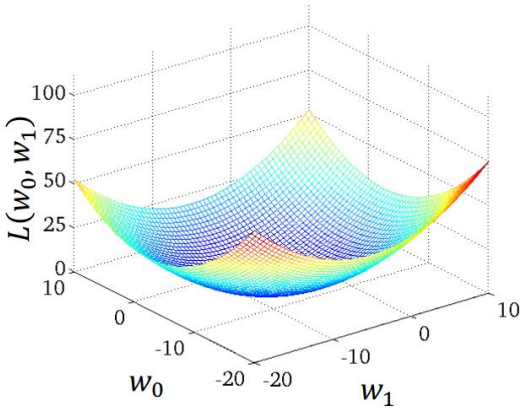
Loss surface

- 다음과 같은 loss surface에서 어디가 최적점(optimal point)일까?
- 어떻게 최적점에 도달할 수 있을까?
  - Loss function을 미분해서 최적점을 한번에 찾을 수 있을까?

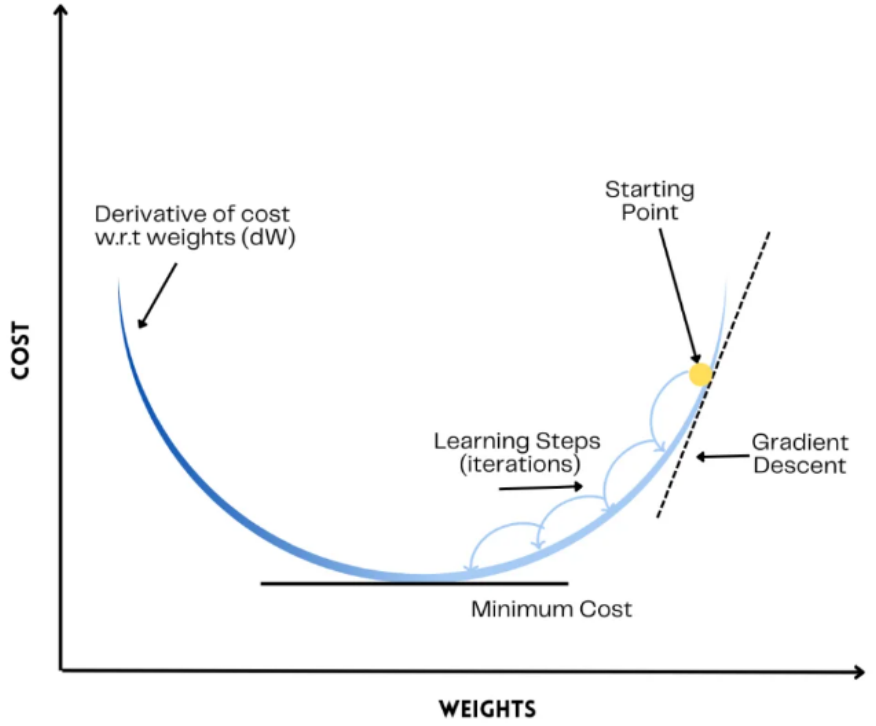


Nonlinear 머신러닝에서 위와 같은 복잡한 loss surface는 자주 등장한다

Gradient Descent Algorithm



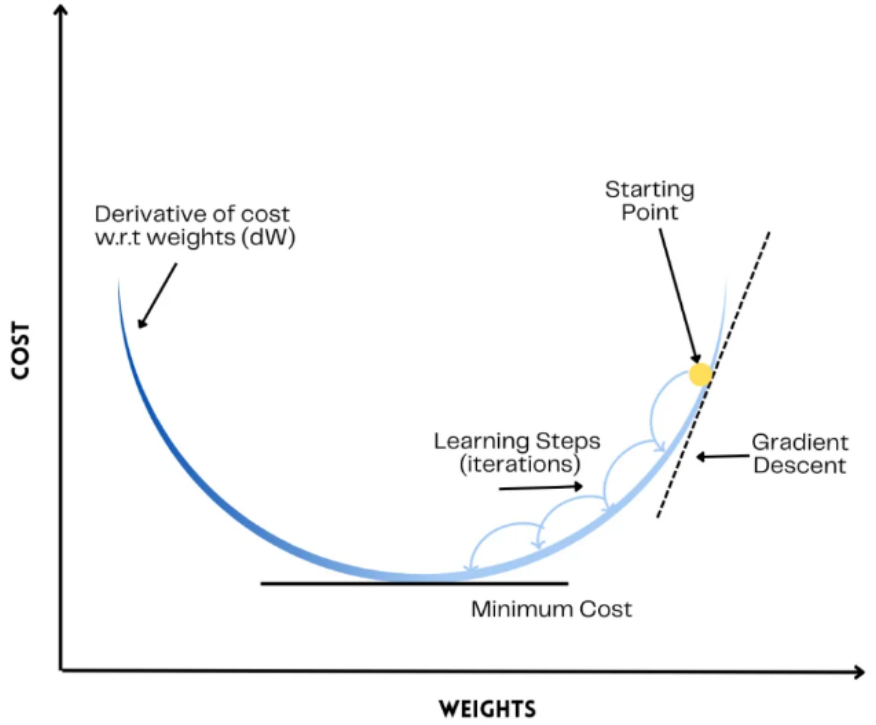
- 머신러닝에서 gradient는 loss function의 미분이다.
- 경사하강법(Gradient Descent algorithm)은 최적점을 찾는 방법 중 하나이다.
  - Gradient의 반대 방향으로 다음 지점을 업데이트한다.
  - Gradient 값이 0이 될 때 까지 가중치(weight; parameter)를 업데이트 한다.



Gradient Descent Algorithm

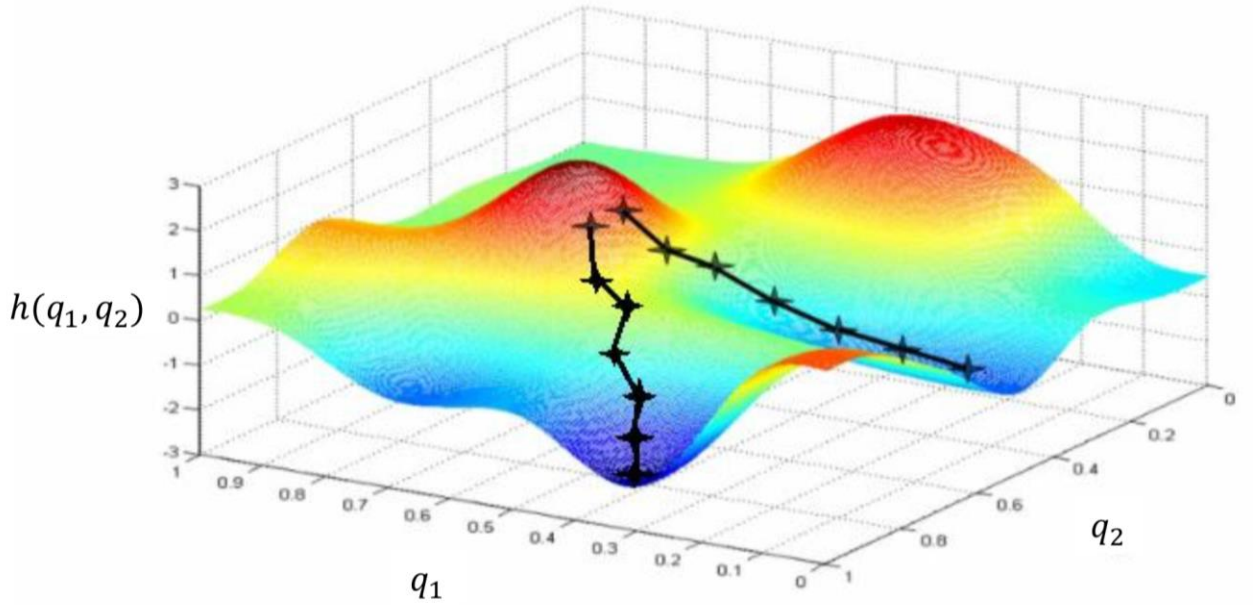
- $w_0 \leftarrow w_0 - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1)$
- $w_1 \leftarrow w_1 - \alpha \frac{\partial}{\partial w_1} L(w_0, w_1)$

α 가 너무 크거나 너무 작으면 어떻게 될까?



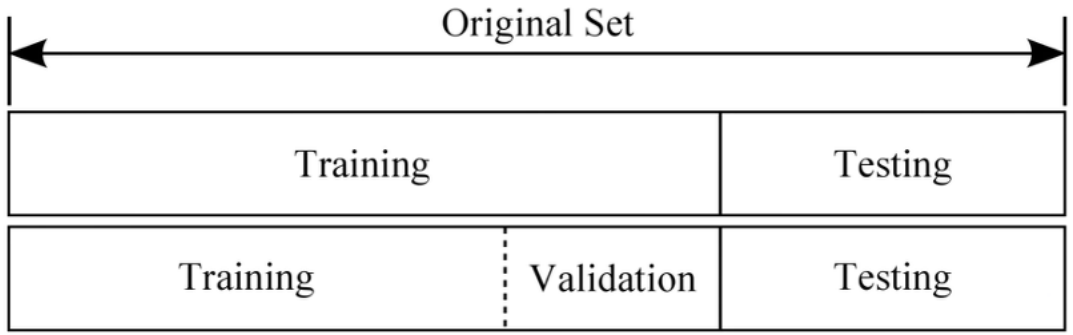
Gradient Descent Algorithm

- 안타깝게도 경사하강법은 완벽하지 않다.
- 경사하강법은 global optimum으로의 수렴을 보장하지 않기 때문이다.
  - 경사하강법은 local optimum으로 수렴한다.
  - Global optimum으로 수렴할 수 없을까?



Train set, Validation set

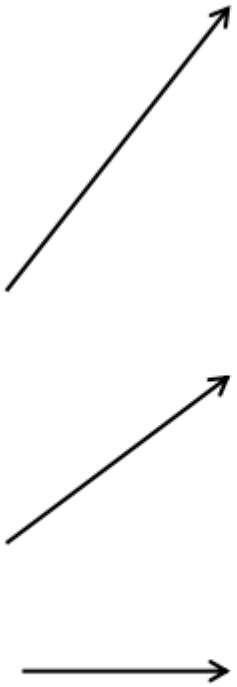
- 모델의 성능을 평가하기 위해서 validation set이 필요하다.
  - 만약 모델 성능 평가를 하지 않는다면 우리의 모델을 평가할 방법이 없다.
- 모델의 목적은 train set에서 잘하는 것이 아니라, test set에서 잘하는 것이다.
  - 하지만 test set은 내가 가지지 못한 데이터셋이다.
- 따라서 내가 갖고 있는 데이터셋의 일부를 validation set으로 두고 나머지에 대해서 학습한다.



Original set은 세상에 존재하는 모든 데이터셋이고, test set은 내가 가지지 못한 데이터셋이다

Train set, Validation set

Size	Price
2,104	400
1,600	330
2,400	369
1,416	232
3,000	540
1,985	300
1,534	315
1,427	199
1,380	212
1,494	243



(x_1, y_1)
(x_2, y_2)
$\vdots$
$(x_{N_{train}}, y_{N_{train}})$

(x_1^{cv}, y_1^{cv})
(x_2^{cv}, y_2^{cv})
$\vdots$
$(x_{N_{cv}}^{cv}, y_{N_{cv}}^{cv})$

(x_1^{test}, y_1^{test})
(x_2^{test}, y_2^{test})
$\vdots$
$(x_{N_{test}}^{test}, y_{N_{test}}^{test})$

- Training error
  - $L(w) = \underset{w}{\text{argmin}} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (y_i - \hat{y}_i)^2$
- Validation error
  - $L(w) = \underset{w}{\text{argmin}} \frac{1}{N_{valid}} \sum_{i=1}^{N_{valid}} (y_i - \hat{y}_i)^2$
- Test error
  - $L(w) = \underset{w}{\text{argmin}} \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (y_i - \hat{y}_i)^2$

K-Fold Cross Validation

Size	Price
2,104	400
1,600	330
2,400	369
1,416	232
3,000	540
1,985	300
1,534	315
1,427	199
1,380	212
1,494	243

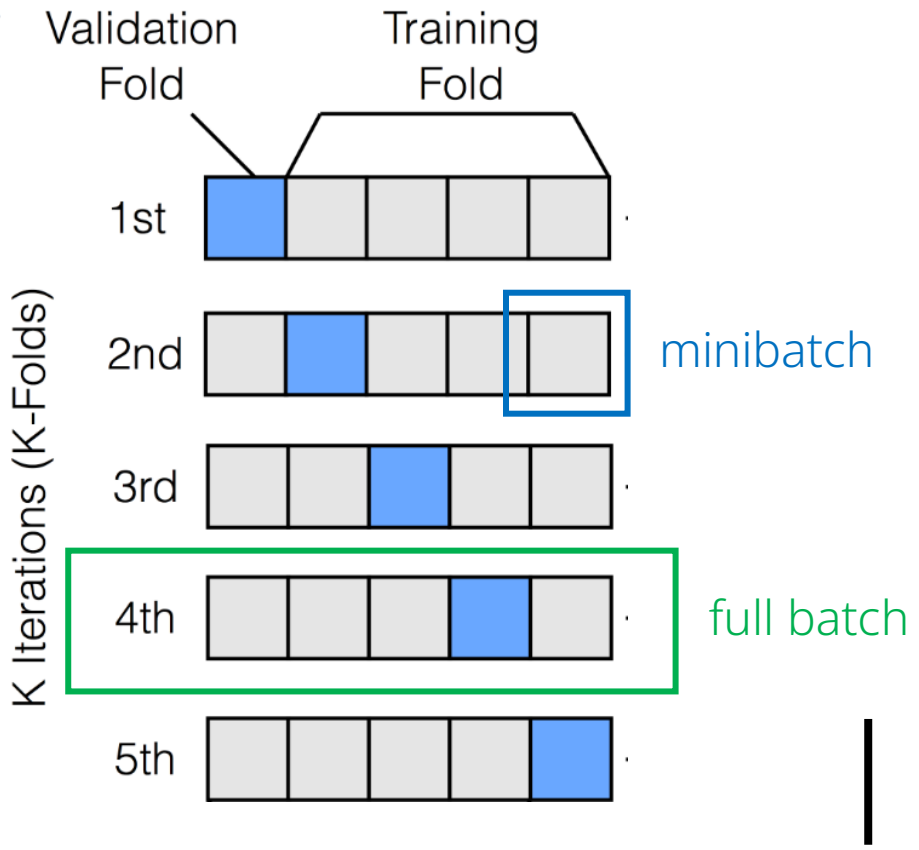
Size	Price
2,104	400
1,600	330
2,400	369
1,416	232
3,000	540
1,985	300
1,534	315
1,427	199
1,380	212
1,494	243

...

Size	Price
2,104	400
1,600	330
2,400	369
1,416	232
3,000	540
1,985	300
1,534	315
1,427	199
1,380	212
1,494	243

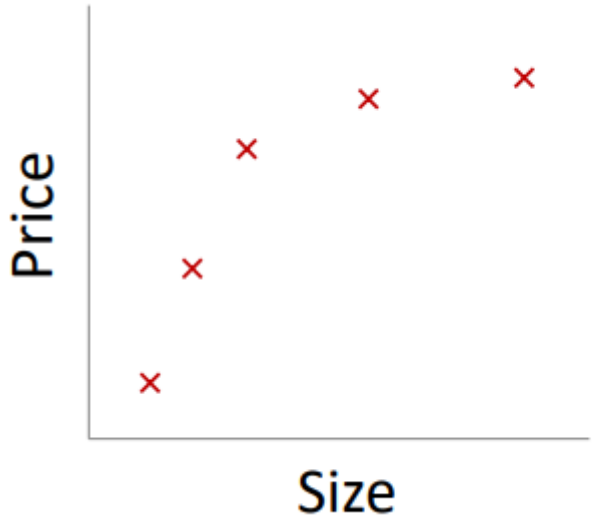
K-Fold Cross Validation

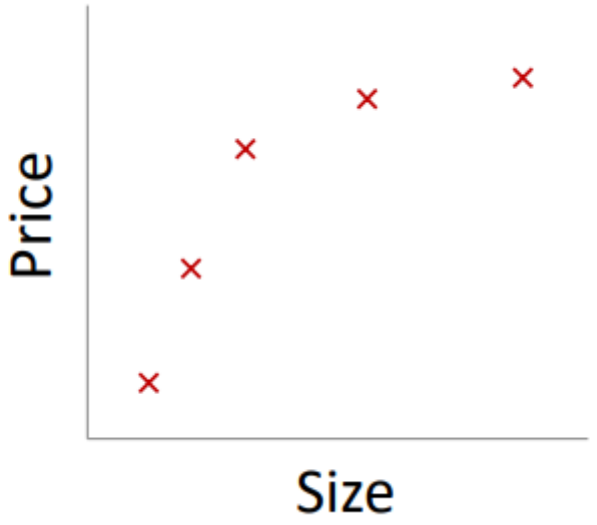
- 데이터셋을 k개로 분할하여 매번 validation set을 달리하는 방법을 K-Fold Cross Validation이라고 한다.
  - validation set을 제외한 다른 데이터셋은 train set으로 모델 훈련에 사용한다.
- 분할된 1개의 조그만 데이터셋을 minibatch라 한다.
- 전체 데이터셋을 (full) batch라 한다.
- 모델이 full batch를 한 번 학습했을 때, 이를 1 epoch라고 한다.

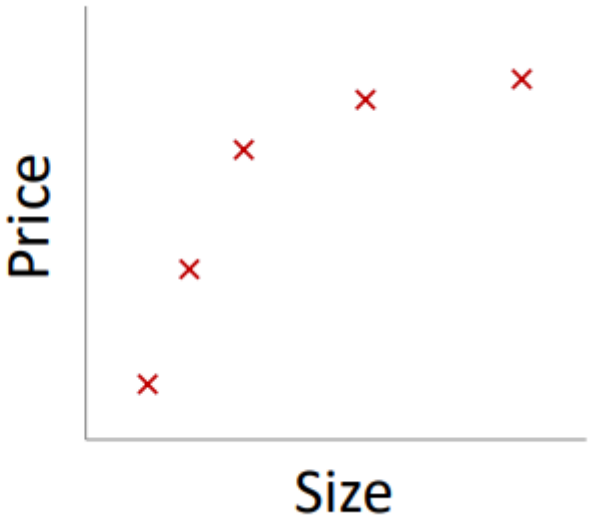


Overfitting & Underfitting

- 아래 회귀 문제에서 각각 underfitting, well-fit, overfitting 상황을 그려보자.

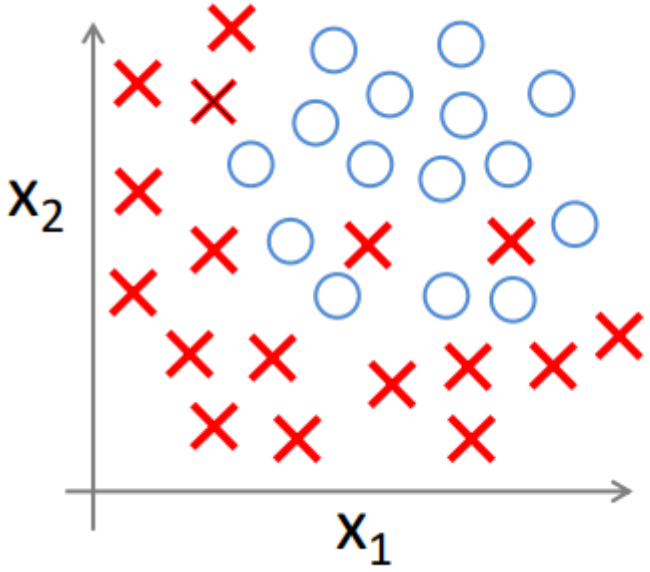


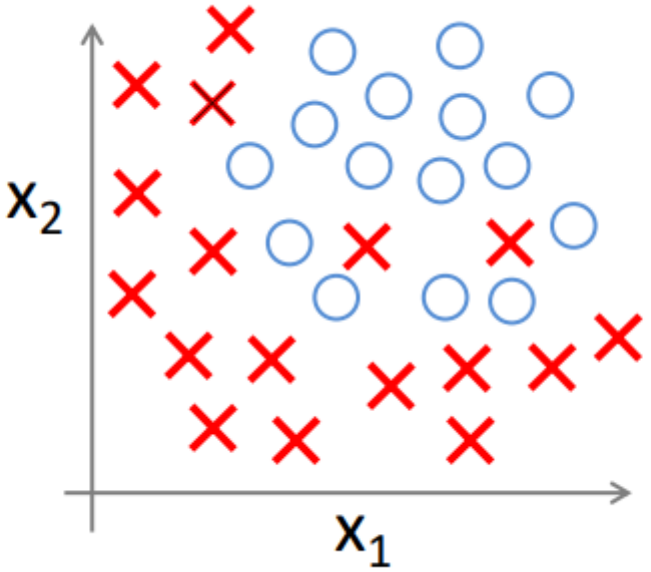


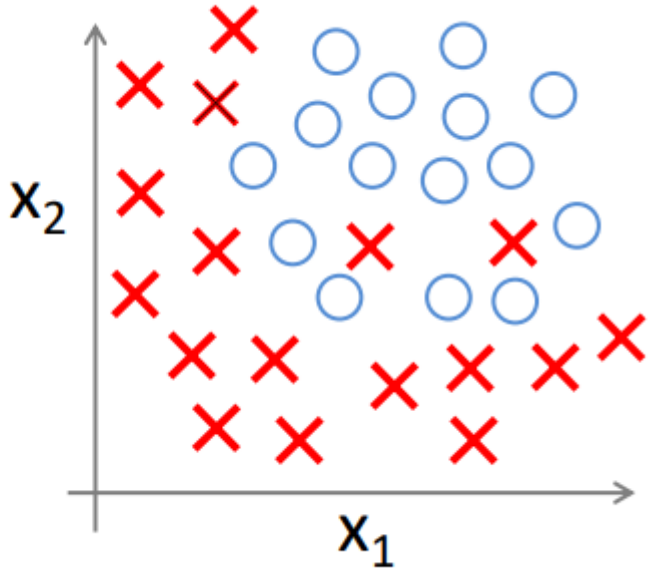


Overfitting & Underfitting

- 아래 이진 분류 문제에서 각각 underfitting, well-fit, overfitting 상황을 그려보자.

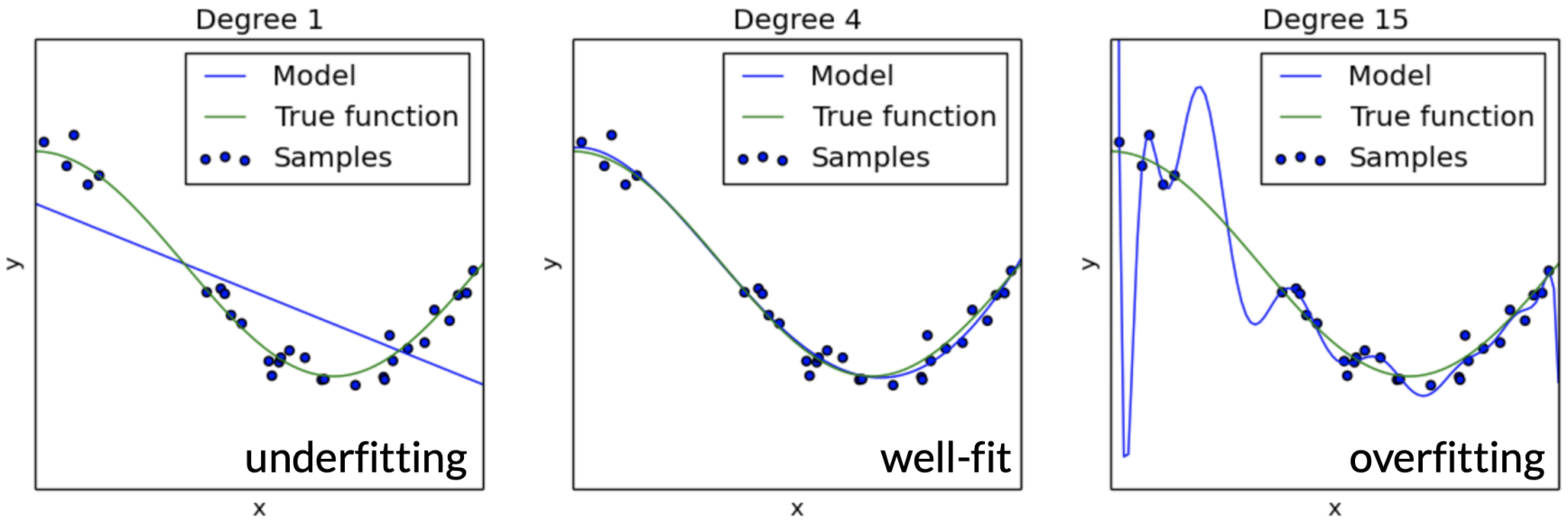






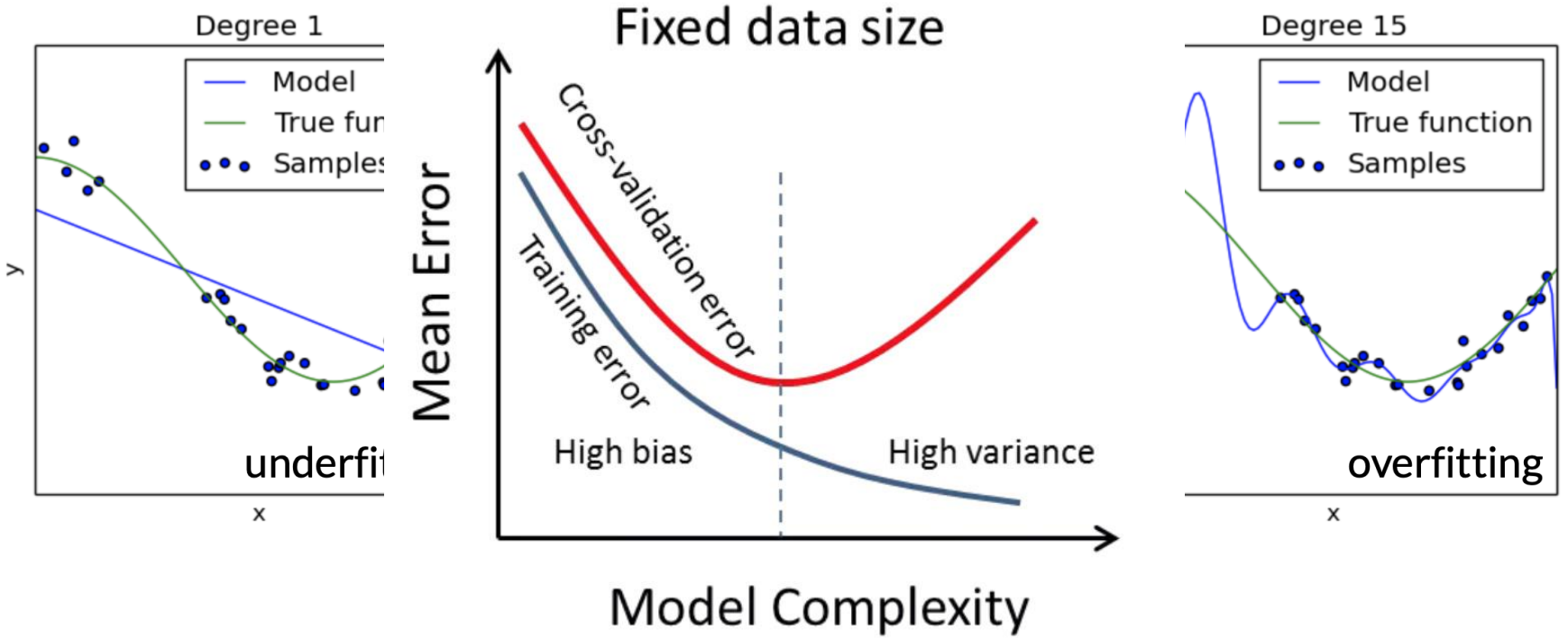
Overfitting & Underfitting

- Overfitting이나 Underfitting은 우리 모델의 일반화(generalization) 능력이 떨어짐을 의미한다.



Overfitting & Underfitting

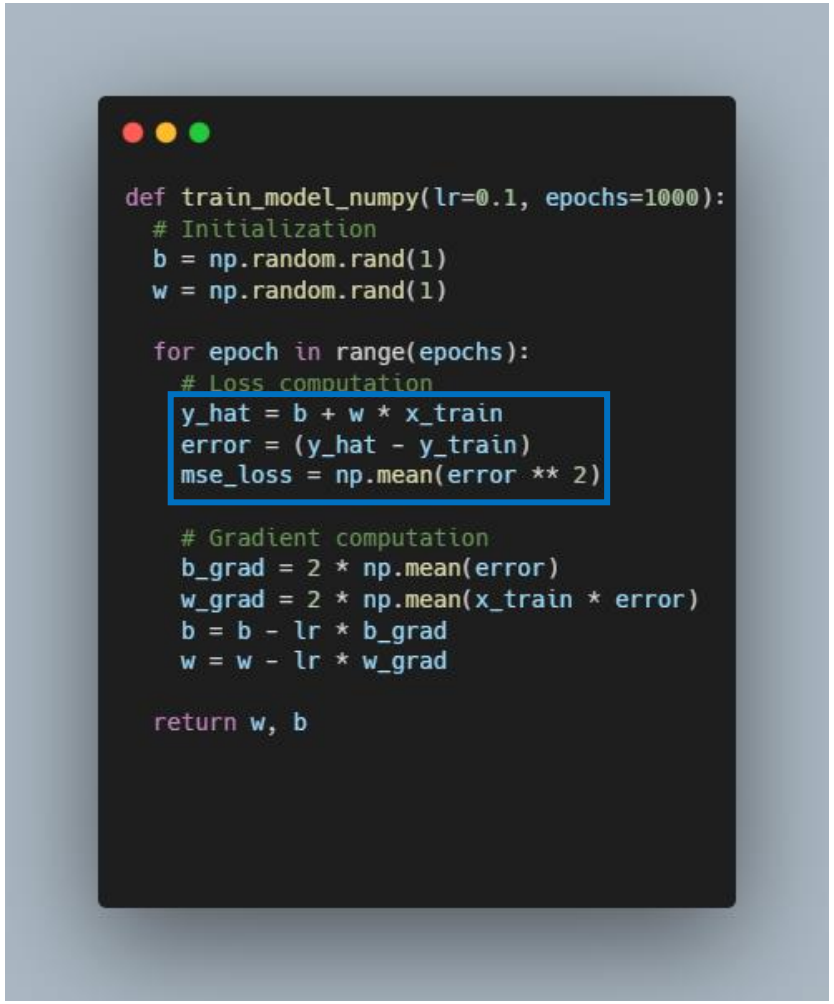
- Overfitting이나 Underfitting은 우리 모델의 일반화(generalization) 능력이 떨어짐을 의미한다.



From now on, let’s practice!

- 다음 사이트에 접속해서 실습을 진행합니다: https://colab.research.google.com/?hl=ko
- Google Colab에서 실습 코드를 돌릴 수 있습니다.
- Google Colab은 구글이 제공하는 외부 GPU를 사용하는 방식입니다.
- Google Colab 유료버전을 사용해도 좋습니다.
- GPU를 너무 오랫동안 사용하면 무료 버전 credit이 소진되기도 합니다. 이 때는 부계정을 이용하세요.
- 노트북에 GPU가 있다면 .py로 코딩해도 괜찮지만 CUDA 등 환경설정을 해본 적 없다면 Google Colab으로 실습하세요.
- 실습 코드 및 팀 퀴즈는 Github에 올리고 강의자를 Collaborator로 등록합니다.
- Github 이슈는 pacemaker와 함께 해결합니다.

Gradient Descent for Linear Regression



```python
def train_model_numpy(lr=0.1, epochs=1000):
  # Initialization
  b = np.random.rand(1)
  w = np.random.rand(1)

  for epoch in range(epochs):
    # Loss computation
    y_hat = b + w * x_train
    error = (y_hat - y_train)
    mse_loss = np.mean(error ** 2)

    # Gradient computation
    b_grad = 2 * np.mean(error)
    w_grad = 2 * np.mean(x_train * error)
    b = b - lr * b_grad
    w = w - lr * w_grad

  return w, b
```

Gradient Descent for Linear Regression

```python
def train_model_numpy(lr=0.1, epochs=1000):
  # Initialization
  b = np.random.rand(1)
  w = np.random.rand(1)

  for epoch in range(epochs):
    # Loss computation
    y_hat = b + w * x_train
    error = (y_hat - y_train)
    mse_loss = np.mean(error ** 2)

    # Gradient computation
    b_grad = 2 * np.mean(error)
    w_grad = 2 * np.mean(x_train * error)
    b = b - lr * b_grad
    w = w - lr * w_grad

  return w, b
```

Data Generation with PyTorch

```python
import torch

# create tensor at CPU
x_train_tensor = torch.as_tensor(x_train)
y_train_tensor = torch.as_tensor(y_train)

# create tensor at GPU
device = 'cuda' if torch.cuda.is_available() else 'cpu'
x_train_tensor = torch.as_tensor(x_train).to(device)
y_train_tensor = torch.as_tensor(y_train).to(device)
```

이번에는 NumPy 코드를 PyTorch 코드로 바꿔봅시다!

Gradient Descent by PyTorch

PyTorch

```python
def train_model_torch(lr=0.1, epochs=1000):
  b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
  w = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)

  for epoch in range(epochs):
    # Loss computation
    y_hat = b + w * x_train_tensor
    error = (y_hat - y_train_tensor)
    mse_loss = torch.mean(error ** 2)

    # Gradient computation and descent
    mse_loss.backward()
    with torch.no_grad():
      b -= lr * b.grad
      w -= lr * w.grad
    b.grad.zero_()
    w.grad.zero_()

  return w, b
```

In-place 연산은 no_grad()를 반드시 해주어야 합니다

NumPy

```python
def train_model_numpy(lr=0.1, epochs=1000):
  # Initialization
  b = np.random.rand(1)
  w = np.random.rand(1)

  for epoch in range(epochs):
    # Loss computation
    y_hat = b + w * x_train
    error = (y_hat - y_train)
    mse_loss = np.mean(error ** 2)

    # Gradient computation
    b_grad = 2 * np.mean(error)
    w_grad = 2 * np.mean(x_train * error)
    b = b - lr * b_grad
    w = w - lr * w_grad

  return w, b
```

In-place Operation

- In-place 연산은 copy를 만들지 않고 linear algebra, vector, matrices(Tensor)를 바로 변화시키는 연산을 의미한다.
- In-place 연산이 있을 경우 with torch.no_grad(): 를 이용해 PyTorch가 파라미터들의 gradient를 tracking하는 것을 막는다.

NumPy

```
[In [1]: import numpy as np

[In [2]: x = np.array(1)

[In [3]: y = np.array(2)

[In [4]: id(x), id(y)
Out[4]: (140685031061024, 140685035868528)

[In [5]: x += y

[In [6]: print(x), id(x)
3
Out[6]: (None, 140685031061024)

[In [7]: x = x + y

[In [8]: print(x), id(x)
5
Out[8]: (None, 140685029890896)
```

PyTorch

```
[In [1]: import torch

[In [2]: x = torch.tensor(1)

[In [3]: y = torch.tensor(2)

[In [4]: id(x), id(y)
Out[4]: (140399695057200, 140399395412064)

[In [5]: x += y

[In [6]: print(x), id(x)
tensor(3)
Out[6]: (None, 140399695057200)

[In [7]: x = x + y

[In [8]: print(x), id(x)
tensor(5)
Out[8]: (None, 140399395413584)
```

Gradient Descent by PyTorch

```python
def train_model_torch_optim(lr=0.1, epochs=1000):
  b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
  w = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)

  parameters = [b, w]
  optimizer = optim.SGD(parameters, lr=lr)
  mse_loss = nn.MSELoss()

  for epoch in range(epochs):
    # Loss computation
    y_hat = b + w * x_train_tensor
    loss = mse_loss(y_hat, y_train_tensor)

    # Gradient computation and descent
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

  return w, b
```

자주 사용되는 코드이니 외워둡시다!

SGD가 무엇인지 다음 시간에 배울 예정입니다.

Gradient Descent by PyTorch

```python
def train_model_torch_optim(lr=0.1, epochs=1000):
  b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
  w = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)

  parameters = [b, w]
  optimizer = optim.SGD(parameters, lr=lr)
  mse_loss = nn.MSELoss()

  for epoch in range(epochs):
    # Loss computation
    y_hat = b + w * x_train_tensor
    loss = mse_loss(y_hat, y_train_tensor)

    # Gradient computation and descent
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

  return w, b
```

연산을 정의하면 PyTorch가 알아서

역전파 계산 등을 해줍니다!

Quiz #1

- PyTorch로 추정한 parameter를 이용해서 test 데이터에서 MSE error를 계산하는 코드 구현
  - 필수조건
    - NumPy 코드 사용 금지
    - GPU device 사용하기

- Github repo에 코드를 올려두기

Quiz #2

- 제공한 pickle 데이터셋을 fitting하는 함수의 parameter 추정
  - 필수조건
    - 추가 라이브러리 사용 금지
    - 10회 반복 측정한 평균 test MSE error 값 0.05미만
  - 도전
    - 학습시간 500ms(=0.5초) 미만

- Github repo에 코드를 올려두기

KUGODS